Using StoreKit for In-App Purchases with Swift 3

Session 702

Dana DuBois  App Store Engineering Manager
What’s New
What’s New

APIs in Swift
What’s New

APIs in Swift
Subscriptions
Subscriptions
Subscriptions

Expanded categories
Subscriptions

Expanded categories
Increased proceeds
Subscriptions

- Expanded categories
- Increased proceeds
- Territory pricing
Subscriptions

Expanded categories
Increased proceeds
Territory pricing
Preserve prices
Subscriptions

Expanded categories
Increased proceeds
Territory pricing
Preserve prices
Upgrades and downgrades
Subscriptions

Expanded categories
Increased proceeds
Territory pricing
Preserve prices
Upgrades and downgrades

Introducing Expanded Subscriptions in iTunes Connect
What’s New

APIs in Swift
Subscriptions
What’s New

APIs in Swift
Subscriptions
iMessage apps
iMessage Apps
iMessage Apps

iMessage extensions will support In-App Purchases
iMessage Apps

iMessage extensions will support In-App Purchases
Same StoreKit APIs
In-App Purchase Overview
In-App Purchase Overview

Digital content or service bought in app
In-App Purchase Overview

Digital content or service bought in app
Not for physical goods
Types of In-App Purchases
Types of In-App Purchases

Consumable products
Types of In-App Purchases

Consumable products
Non-consumable products
Types of In-App Purchases

Consumable products
Non-consumable products
Non-renewing subscriptions
Types of In-App Purchases

Consumable products
Non-consumable products
Non-renewing subscriptions
Auto-renewing subscriptions
Implementing In-App Purchases
In-App Purchase Process

Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
In-App Purchase Process

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
In-App Purchase Process

Load In-App Identifiers → Fetch Product Info → Show In-App UI → Make Purchase → Process Transaction → Make Asset Available → Finish Transaction
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
In-App Purchase Process

Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction
The Payment Queue
The Payment Queue

The center of your In-App Purchase implementation
• The only source of truth for state
The Payment Queue

The center of your In-App Purchase implementation

• The only source of truth for state

Rely on the queue, and only the queue

• For transactions in progress
• Payment status updates
• Download status
The Payment Queue

The center of your In-App Purchase implementation

• The only source of truth for state

Rely on the queue, and only the queue

• For transactions in progress
• Payment status updates
• Download status

Any and all transactions in the queue are valid and real
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
}
import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
}
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
}
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.defaultQueue().add(self);
        return true
    }
}
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
Load In-App Identifiers
Load In-App Identifiers

Options for storing the list of product identifiers
Load In-App Identifiers

Options for storing the list of product identifiers

- Baked-in product identifier
Load In-App Identifiers

Options for storing the list of product identifiers

- Baked-in product identifier
- Fetch from server
In-App Purchase Process

- Load In-App Identifiers
- Fetch Product Info
- Show In-App UI
- Make Purchase
- Process Transaction
- Make Asset Available
- Finish Transaction
let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {

    for product in response.products {

        // Localized title and description
        product.localizedTitle
        product.localizedDescription

        // Price and locale
        product.price
        product.priceLocale

        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }

}
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription

        // Price and locale
        product.price
        product.priceLocale

        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse)
{
    for product in response.products {
        // Localized title and description
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle Events

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
            case .purchased:
                // Validate the purchase

            case .deferred:
                // Allow the user to continue to use the app
                // It may be some time before the transaction is updated
                // Do not get stuck in a modal “Purchasing...” state!
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase

        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal “Purchasing...” state!

        }
    }
}
Testing Deferred Transactions
Testing Deferred Transactions

Create a mutable payment
Testing Deferred Transactions

Create a mutable payment
Set the simulatesAskToBuyInSandbox flag
Testing Deferred Transactions

Create a mutable payment
Set the simulatesAskToBuyInSandbox flag

```swift
let payment = SKMutablePayment(product: product)
payment.simulatesAskToBuyInSandbox = true
SKPaymentQueue.defaultQueue().add(payment)
```
Handling Errors
Handling Errors

Not all errors are equal
Handling Errors

Not all errors are equal

Check the error code
- Don’t show an error alert unless necessary
- User canceling a payment will result in an error
Handling Errors

Not all errors are equal

Check the error code
- Don’t show an error alert unless necessary
- User canceling a payment will result in an error

Let StoreKit handle the transaction flow as much as possible
- Including asking for confirmation for purchase
Validate the Purchase

Working with receipts
Receipt Validation
Receipt Validation

On-device validation

• Unlock features and content within the app
Receipt Validation

On-device validation
• Unlock features and content within the app

Server-to-server validation
• Restrict access to downloadable content
Receipt Validation

On-device validation
• Unlock features and content within the app

Server-to-server validation
• Restrict access to downloadable content

Do not use online validation directly from the device!
The Receipt
The Receipt

Trusted record of App and In-App Purchases
The Receipt

Trusted record of App and In-App Purchases

Stored on device
The Receipt

Trusted record of App and In-App Purchases
Stored on device
Issued by the App Store
The Receipt

Trusted record of App and In-App Purchases
Stored on device
Issued by the App Store
Signed and verifiable
The Receipt

Trusted record of App and In-App Purchases
Stored on device
Issued by the App Store
Signed and verifiable
For your app, on that device only
The Basics

Receipt

Purchase Information

Certificates

Signature
The Basics

Stored in the App Bundle

• API to get the path
The Basics

Stored in the App Bundle
• API to get the path

Single file
• Purchase data
• Signature to check authenticity

[Diagram: Receipt, Purchase Information, Certificates, Signature]
Standards

Receipt

Purchase Information

Certificates

Signature
Standards

Signing

• PKCS#7 Cryptographic Container
Standards

Signing
• PKCS#7 Cryptographic Container

Data Encoding
• ASN.1

Receipt

Purchase Information

Certificates

Signature
Standards

Signing
• PKCS#7 Cryptographic Container

Data Encoding
• ASN.1

Options for verifying and reading
• OpenSSL, asn1c, etc.
• Create your own
Getting Started
Getting Started

Locate the receipt using NSBundle API
Locate the receipt using NSBundle API

// Locate the file
let url = NSBundle.main().appStoreReceiptURL!

// Read the contents
let receipt = NSData(contentsOf: url)
Verification
Verification

Do not check the expiry date on the certificate
Verification

Do not check the expiry date on the certificate

Do evaluate trust up to Root CA
Receipt Payload

Receipt

Purchase Information

Attribute
Type 2
Bundle Identifier

Attribute
Type #
Value

Attribute
Type #
Value

Certificates

Signature
Receipt Payload

Series of attributes

- Type
- Value
- (Version)
Verify Application
Verify Application

Check the Bundle Identifier

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
<td>Bundle Identifier</td>
</tr>
<tr>
<td>Type 3</td>
<td>Bundle Version</td>
</tr>
</tbody>
</table>
Verify Application

Check the Bundle Identifier

Check the Bundle Version
Verify Application

Check the Bundle Identifier

Check the Bundle Version

Use hardcoded values
  • Not Info.plist values

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type 2</th>
<th>Bundle Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type 3</td>
<td>Bundle Version</td>
</tr>
</tbody>
</table>
Verify Device

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle Identifier</td>
<td>Type 2</td>
</tr>
<tr>
<td>Bundle Version</td>
<td>Type 3</td>
</tr>
<tr>
<td>Opaque Value</td>
<td>Type 4</td>
</tr>
<tr>
<td>SHA-1 Hash</td>
<td>Type 5</td>
</tr>
</tbody>
</table>
Verify Device

Attribute 5 is a SHA-1 hash of 3 key values
- Bundle ID
- Device Identifier
- Opaque Value

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
<td>Bundle Identifier</td>
</tr>
<tr>
<td>Type 3</td>
<td>Bundle Version</td>
</tr>
<tr>
<td>Type 4</td>
<td>Opaque Value</td>
</tr>
<tr>
<td>Type 5</td>
<td>SHA-1 Hash</td>
</tr>
</tbody>
</table>
Verify Device

Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

The App Store knows these at time of purchase
Verify Device

Attribute 5 is a SHA-1 hash of 3 key values

- Bundle ID
- Device Identifier
- Opaque Value

The App Store knows these at time of purchase

Your app knows them at time of verification
Verify Device

Attribute 5 is a SHA-1 hash of 3 key values
• Bundle ID
• Device Identifier
• Opaque Value

The App Store knows these at time of purchase

Your app knows them at time of verification

Unique to your app on this device
In-App Purchase Attributes

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
<td>Bundle Identifier</td>
</tr>
</tbody>
</table>
In-App Purchase Attributes

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type 2</th>
<th>Bundle Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>
In-App Purchase Attributes

- Attribute 5 is a SHA-1 hash of 3 key values
  - Your app knows them at time of verification

Receipt

- Purchase Information
  - Attribute
    - Type 2: Bundle Identifier
  - ... attribute
  - Attribute
    - Type 17: In-App Purchase
  - Attribute
    - Type 17: In-App Purchase
  - Attribute
    - Type 17: In-App Purchase
In-App Purchase Attributes

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Purchase Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
<td>Bundle Identifier</td>
</tr>
<tr>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type 1701</th>
<th>Quantity</th>
</tr>
</thead>
</table>
In-App Purchase Attributes

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
</tr>
<tr>
<td>Bundle Identifier</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type 1701</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1702</td>
<td>Product Identifier</td>
</tr>
</tbody>
</table>
In-App Purchase Attributes

Receipt

Purchase Information

Attribute
Type 2
Bundle Identifier

Attribute
Type 17
In-App Purchase

Attribute
Type 17
In-App Purchase

Attribute
Type 17
In-App Purchase

In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type 1701</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1702</td>
<td>Product Identifier</td>
</tr>
<tr>
<td>Type 1703</td>
<td>Transaction Identifier</td>
</tr>
</tbody>
</table>
### In-App Purchase Attributes

**Receipt**

**Purchase Information**

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
</tr>
<tr>
<td>Bundle Identifier</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

### In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type 1701</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1702</td>
<td>Product Identifier</td>
</tr>
<tr>
<td>Type 1703</td>
<td>Transaction Identifier</td>
</tr>
<tr>
<td>Type 1704</td>
<td>Purchase Date</td>
</tr>
</tbody>
</table>
In-App Purchase Attributes

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type 2</th>
<th>Bundle Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type 1701</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1702</td>
<td>Product Identifier</td>
</tr>
<tr>
<td>Type 1703</td>
<td>Transaction Identifier</td>
</tr>
<tr>
<td>Type 1704</td>
<td>Purchase Date</td>
</tr>
<tr>
<td>Type 1708</td>
<td>Subscription Expiration Date</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Switching to Subscriptions

Receipt

Purchase Information

Attribute

Type 19
Original Application Version
Switching to Subscriptions

Original application version in the receipt

Receipt

Purchase Information

Attribute

Type 19  Original Application Version
Switching to Subscriptions

Original application version in the receipt
Know whether to treat the app as the paid version, or the subscription version
Transaction Lifecycle
Transaction Lifecycle

Consumable and non-renewing subscriptions

- Will only appear once
- In the receipt issued at time of purchase
- Will not be present in subsequent receipts issued
Transaction Lifecycle

Consumable and non-renewing subscriptions
• Will only appear once
• In the receipt issued at time of purchase
• Will not be present in subsequent receipts issued

Non-consumable and auto-renewable subscriptions
• Always in the receipt
• Can be restored via StoreKit API
Receipt Refresh on iOS
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit
Receipt refresh will require network
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit
Receipt refresh will require network
Store sign-in will be required
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit
Receipt refresh will require network
Store sign-in will be required
Avoid continuous loop of validate-and-refresh
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, Refresh the receipt using StoreKit
Receipt refresh will require network
Store sign-in will be required
Avoid continuous loop of validate-and-refresh

```swift
let request = SKReceiptRefreshRequest()
request.delegate = self;
request.start()
```
Receipt Refresh on macOS
Receipt Refresh on macOS

If the receipt is invalid
Receipt Refresh on macOS

If the receipt is invalid
Exit with code 173 to refresh receipt
Receipt Refresh on macOS

If the receipt is invalid
Exit with code 173 to refresh receipt
Receipt refresh will require network
Receipt Refresh on macOS

If the receipt is invalid
Exit with code 173 to refresh receipt
Receipt refresh will require network
Store sign-in will be required
Receipt Refresh on macOS

If the receipt is invalid
Exit with code 173 to refresh receipt
Receipt refresh will require network
Store sign-in will be required

// Receipt is invalid
exit(173);
Server-to-Server Validation
Server-to-Server Validation

Allows your servers to validate the receipt before issuing content
Server-to-Server Validation

Allows your servers to validate the receipt before issuing content

Your app sends the receipt to your servers
• Your server sends the receipt to Apple’s server
• Never send the receipt directly from your app to Apple’s server
Server-to-Server Validation

Allows your servers to validate the receipt before issuing content

Your app sends the receipt to your servers

- Your server sends the receipt to Apple’s server
- Never send the receipt directly from your app to Apple’s server

Response is in JSON
In-App Purchase Process

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Make Purchase
5. Process Transaction
6. Make Asset Available
7. Finish Transaction
Make Asset Available
Make Asset Available

Unlock functionality in your app
Make Asset Available

Unlock functionality in your app
Download additional content
On-Demand Resources
On-Demand Resources

Hosted on the App Store
On-Demand Resources

Hosted on the App Store

Can contain any data type except executable Swift, Objective-C, C, or C++ code
On-Demand Resources

Hosted on the App Store
Can contain any data type except executable Swift, Objective-C, C, or C++ code
Available on iOS and tvOS
On-Demand Resources

Hosted on the App Store
Can contain any data type except executable Swift, Objective-C, C, or C++ code
Available on iOS and tvOS

Optimizing On-Demand Resources

Mission
Thursday 10:00AM
Hosted In-App Purchase Content
Hosted In-App Purchase Content

Hosted on Apple’s servers
Hosted In-App Purchase Content

Hosted on Apple’s servers

Scalable and reliable
Hosted In-App Purchase Content

Hosted on Apple’s servers
Scalable and reliable
Downloads in background
Hosted In-App Purchase Content

Hosted on Apple’s servers
Scalable and reliable
Downloads in background
Up to 2GB per in-app purchasable product
Hosted In-App Purchase Content

Hosted on Apple’s servers
Scalable and reliable
Downloads in background
Up to 2GB per in-app purchasable product
Supported on iOS, tvOS, and macOS
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions [SKPaymentTransaction]) {

    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions [SKPaymentTransaction]) {
    for transaction in transactions {
        if transaction.downloads.count > 0 {
            SKPaymentQueue.defaultQueue().start(transaction.downloads)
        }
    }
}
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {

    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
// Hosted Content

func paymentQueue(_ queue: SKPaymentQueue, updatedDownloads downloads: [SKDownload]) {
    for download in downloads {
        download.progress
        download.timeRemaining
        download.error

        if download.downloadState == .finished {
            download.contentURL
        }
    }
}
Self-Hosted Content
Self-Hosted Content

Use background download APIs
Self-Hosted Content

Use background download APIs

- Content is downloaded even when your app is not active
Self-Hosted Content

Use background download APIs

• Content is downloaded even when your app is not active
• NSURLConnection is deprecated
let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier: "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier: "MyBackgroundSession")

let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
let config = URLSessionConfiguration.backgroundSessionConfiguration(withIdentifier: "MyBackgroundSession")
let session = URLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = URLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
let config = URLSessionConfiguration.backgroundSessionConfiguration(withIdentifier: "MyBackgroundSession")
let session = URLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
let config = NSURLSessionConfiguration.backgroundSessionConfiguration(withIdentifier: "MyBackgroundSession")
let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
let request = NSURLRequest(url: myURL)
let downloadTask = session.downloadTask(with: request)
downloadTask.resume()
func URLSession(_ session: URLSession, 
  downloadTask: URLSessionDownloadTask, 
  didWriteData bytesWritten: Int64, 
  totalBytesWritten: Int64, 
  totalBytesExpectedToWrite: Int64) {

  // Do something with progress
}

// Self-Hosted Content
func urlSession(_ session: NSURLSession,
    downloadTask: NSURLSessionDownloadTask,
    didWriteData bytesWritten: Int64,
    totalBytesWritten: Int64,
    totalBytesExpectedToWrite: Int64) {

    // Do something with progress
}
// Self-Hosted Content

func application(_ application: UIApplication,
    handleEventsForBackgroundURLSession identifier: String,
    completionHandler: () -> Void) {

    let config = NSURLSessionConfiguration.
    backgroundSessionConfiguration(withIdentifier: identifier)

    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)

    self.completionHandler = completionHandler // call when done

}
func application(_ application: UIApplication,
        handleEventsForBackgroundURLSession identifier: String,
        completionHandler: () -> Void) {

    let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)

    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)

    self.completionHandler = completionHandler // call when done

}
func application(_ application: UIApplication,
handleEventsForBackgroundURLSession identifier: String,
completionHandler: () -> Void) {
    let config = NSURLSessionConfiguration.
backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
func application(_ application: UIApplication,
    handleEventsForBackgroundURLSession identifier: String,
    completionHandler: () -> Void) {

    let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)

    self.completionHandler = completionHandler // call when done
}

// Self-Hosted Content
func application(_ application: UIApplication, 
    handleEventsForBackgroundURLSession identifier: String, 
    completionHandler: () -> Void) {

    let config = NSURLSessionConfiguration.
        backgroundSessionConfiguration(withIdentifier: identifier)
    let session = NSURLSession(configuration: config, delegate: self, delegateQueue: queue)
    self.completionHandler = completionHandler // call when done
}
In-App Purchase Process

Load In-App Identifiers → Fetch Product Info → Show In-App UI → Make Purchase → Process Transaction → Make Asset Available → Finish Transaction
Finish the Transaction
Finish the Transaction

When the content is downloaded, finish the transaction
Finish the Transaction

When the content is downloaded, finish the transaction

• Otherwise, the payment will stay in the queue
Finish the Transaction

When the content is downloaded, finish the transaction

• Otherwise, the payment will stay in the queue

• If downloading Apple-hosted content, wait until after the download completes
Finish the Transaction

When the content is downloaded, finish the transaction

- Otherwise, the payment will stay in the queue
- If downloading Apple-hosted content, wait until after the download completes

`SKPaymentQueue.defaultQueue().finishTransaction(transaction)`
Restore Completed Transactions
Restore Completed Transactions

Restoring transactions allows the user to restore

• Non-consumable in-app purchases
• Auto-renewing subscriptions
Restore Completed Transactions

Restoring transactions allows the user to restore
• Non-consumable in-app purchases
• Auto-renewing subscriptions

Consumables and non-renewable subscriptions
• You must persist the state!
Restore Completed Transactions
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
Restore Completed Transactions

```swift
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue
Restore Completed Transactions

```swift
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue

```swift
func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {...}
```

```swift
func paymentQueue(_ queue: SKPaymentQueue,
                      restoreCompletedTransactionsFailedWithError error: NSError) {...}
```
Restore Completed Transactions

```swift
SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
```

Observe the queue

```swift
func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {...}
```

```swift
func paymentQueue(_ queue: SKPaymentQueue,
    restoreCompletedTransactionsFailedWithError error: NSError) {...}
```

Inspect the receipt and unlock content and features accordingly
Tips for Passing App Review
Restore Button
You must have a Restore button
Restore Button

You must have a Restore button

Should be used only for

• Non-consumables
• Auto-renewable subscriptions
Restore Button

You must have a Restore button

Should be used only for
• Non-consumables
• Auto-renewable subscriptions

Restore and Purchase should be separate buttons
Auto-Renewable Subscriptions
Auto-Renewable Subscriptions

You must indicate a privacy policy URL
Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-renewable subscription must be in marketing text
Auto-Renewable Subscriptions

You must indicate a privacy policy URL

Auto-renewable subscription must be in marketing text

After subscribing, the latest issue must become downloadable
Auto-Renewable Subscriptions

You must indicate a privacy policy URL
Auto-renewable subscription must be in marketing text
After subscribing, the latest issue must become downloadable
Paid subscription must provide non-free content
Non-Renewing Subscriptions
Non-Renewing Subscriptions

Asking users to register should be optional
• Unless you offer account-based features
Purchases
Purchases must work!
Summary
Summary

Always observe the Payment Queue
Summary

Always observe the Payment Queue

Fetch localized product information from the App Store
Summary

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Summary

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Use the receipt to validate your purchases
Summary

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Use the receipt to validate your purchases
Make the content available
Summary

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Use the receipt to validate your purchases
Make the content available
Finish the transaction
Summary

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Use the receipt to validate your purchases
Make the content available
Finish the transaction
Allow the user to restore complete transactions
More Information

https://developer.apple.com/wwdc16/702
# Related Sessions

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Expanded Subscriptions in iTunes Connect</td>
<td>Pacific Heights</td>
<td>Tuesday 4:00PM</td>
</tr>
<tr>
<td>Optimizing On-Demand Resources</td>
<td>Mission</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>Lab Name</td>
<td>Room Name</td>
<td>Date/Time</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>In-App Purchase/Subscriptions Lab 1</td>
<td>Frameworks Lab B</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>In-App Purchase/Subscriptions Lab 2</td>
<td>Graphics, Games, and Media Lab A</td>
<td>Friday 9:00AM</td>
</tr>
</tbody>
</table>